LR-PARSING DERIVED

Hesselink, Wim H.

Published in:
Science of computer programming

Publication date:
1992

Science of Computer Programming 19 (1992) 171–196
Elsevier

LR-parsing derived

Wim H. Hesselink

Rijksuniversiteit Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, Netherlands

Communicated by M. Rem
Received June 1989
Revised March 1991

Abstract

Hesselink, W.H., LR-parsing derived, Science of Computer Programming 19 (1992) 171–196.

The LR(k)-parsing algorithm is derived, i.e., presented and proved as an interplay between program development and parsing theory. The program development uses invariants and the new concept of weakest angelic precondition. The parsing theory involved relates rightmost derivability to three other transitive relations on strings. The usual stack of item sets and the finite automaton appear in an optimisation of the abstract algorithm.

0. Introduction

The purpose of this note is to derive and prove the LR-parsing algorithm of Knuth (cf. [7]). The reader is advised to forget momentarily most of what he or she might know of this algorithm. We do not start from scratch completely. The reader is supposed to know context-free grammars and languages, and rightmost derivations. The major prerequisite for reading this note is an acquaintance with predicate calculus and a tendency towards formal verification.

The main argument can be summarised as follows. For any context-free grammar we derive a nondeterministic algorithm that may accept any sentence of the language and cannot accept other strings. If the grammar satisfies the LR(k) condition, the algorithm is deterministic and accepts all sentences of the language.

The structure of the paper is as follows. In Section 1 we compare the angelic nondeterminacy of language theory with the demonic nondeterminacy usually associated with program correctness. In Section 2 we fix some notations. A first approximation of the parsing algorithm is developed in Section 3.

Correspondence to: W.H. Hesselink, Rijksuniversiteit Groningen, Department of Computing Science, P.O. Box 800, 9700 AV Groningen, Netherlands. E-mail: wim@cs.rug.nl.

Section 4 forms the heart of the paper. We introduce a "heuristic" predicate to guide the nondeterminacy by means of k look-ahead symbols. This predicate is used to derive an algorithm in which certain sets of strings are needed. Subsequently, we derive a recurrence equation for these sets. The sets are used in algorithm $P2$, which is proved here to be conditionally correct in the sense that it can only terminate for sentences of the language.

In Section 5 we investigate the relationship between the heuristic predicate and rightmost derivations. The results are used in Section 6 to prove that under angelic nondeterminacy any sentence of the language can be accepted by program $P2$. Section 7 contains the traditional LR(k)-parsing algorithm, called $P4$, that uses a finite automaton and a stack of item sets. It is obtained as an optimisation of program $P2$.

In Section 8 we reach the conclusion: if the grammar satisfies the LR(k) condition, algorithm $P4$ accepts the language. It is also shown that an easy modification of the algorithm generates an error message if the input string does not belong to the language.

Section 9 is devoted to the determination of a set-valued function *Rks* that is introduced in Section 4. Here the complications appear that are due to the possibility of ε-productions. Finally, in Section 10 we summarise some distinctive features of our approach in comparison with Knuth's paper [7].

Our presentation of LR-parsing is an interplay between parsing theory and program development. The theory could have been separated completely from the program development. We prefer, however, to develop the theory according to the needs of the program. For the ease of the reader, we use the symbol "**Theory**" to indicate the fragments of the parsing theory, and the symbol "**Program**" for the parts of the program development.

1. Nondeterminacy and language acceptors

Program. In language theory and complexity theory, nondeterminacy of some mechanism usually leads to the question of whether the mechanism admits at least one computation with a certain property (cf. [6, p. 19, p. 163]). This point of view is called the *angelic* appreciation of nondeterminacy. In programming methodology one usually asks whether all (or all terminating) computations satisfy a certain property (cf. [1, 7.3; 2; 3, Chapter 7]). This point of view is called the *demonic* appreciation. Here we investigate a possibly nondeterministic algorithm for the parsing of languages. Therefore, we must be careful about the appreciation of nondeterminacy.

If X is a boolean function on the state space, we let $[X]$ denote the proposition that X holds everywhere on the state space. In particular, $[X \Rightarrow Y]$ is the proposition that $\neg X \vee Y$ holds everywhere, i.e. that X is stronger than Y. For a command P we write $wp.P.X$ to denote the weakest precondition such that every computation

of P terminates in a state where X holds. We write $wlp.P.X$ to denote the weakest precondition such that every terminating computation of P terminates in a state where X holds (cf. [3, Chapter 7]). The predicate transformers wp and wlp represent the demonic appreciation of nondeterminacy (and wlp corresponds to conditional correctness).

Let us introduce the weakest angelic precondition wap, by specifying that $wap.P.X$ is the weakest precondition such that P has at least one computation that terminates in a state where X holds. Clearly, P has a computation that terminates in a state where X holds if and only if it is not the case that every terminating computation terminates in a state where $\neg X$ holds. Therefore, we have

(0) $\quad [wap.P.X \equiv \neg wlp.P.(\neg X)].$

If P has at least one computation and if every computation of P terminates in a state where X holds, then P has some computation that terminates in a state where X holds. This means

(1) $\quad [\neg wp.P.false \wedge wp.P.X \Rightarrow wap.P.X].$

In this note we only consider total commands, i.e. commands P that satisfy

(2) $\quad [\neg wp.P.false].$

Therefore, formula (1) simplifies to

(3) $\quad [wp.P.X \Rightarrow wap.P.X].$

We now come to the parsing problem. Recall that a language is a set of strings. A program P is called an *acceptor* of language L if and only if $m \in L$ is a necessary and sufficient precondition for termination of P. Sufficiency means that $m \in L$ implies termination, i.e.

(4) $\quad [m \in L \Rightarrow wp.P.true].$

Necessity means that $m \notin L$ implies nontermination, i.e.

(5) $\quad [m \notin L \Rightarrow wlp.P.false].$

With contraposition we get from (0) that (5) is equivalent to

(6) $\quad [wap.P.true \Rightarrow m \in L].$

In general we do not completely know the language. So, the acceptance problem of context-free languages must be posed in terms of grammars. We only consider context-free grammars. For such a grammar G, we write $L.G$ to denote the language that is generated. The problem is to construct a program $P.G$ dependent of a parameter G such that for every grammar G program $P.G$ is an acceptor of the language $L.G$.

It is not difficult to construct a program $P.G$ such that $m \in L.G$ if and only if $P.G$ has some terminating computation, i.e.

(7) $\quad [m \in L.G \equiv wap.(P.G).true].$

Usually, such a program $P.G$ has a nondeterminacy that reflects the freedom any user of the grammar has to generate arbitrary sentences of the language. Clearly, formula (7) implies (6), but it does not imply (4). Nevertheless, it follows that $P.G$ is an acceptor of $L.G$ if $P.G$ satisfies the additional condition

(8) $\quad [wap.(P.G).true \equiv wp.(P.G).true],$

i.e. that termination of $P.G$ is deterministic. Anyhow, since angelic nondeterminacy is not well implementable, it seems reasonable to impose condition (8). There are two possibilities to get (8) for a parametrised program $P.G$ that satisfies (7). In both cases we restrict the class of grammars. The first possibility is to make the program deterministic by strengthening the guards of the nondeterminate choices. In this way, condition (8) is forced, but (7) may have been invalidated. The modified program is correct for the grammars for which (7) remains valid. These grammars, however, may be difficult to characterise.

The other possibility is to preserve program $P.G$ and to characterise the grammars G for which $P.G$ happens to be deterministic. In this way no new proof obligations appear. This approach has the drawback that in the design of program $P.G$ we have the implicit consideration that the class of grammars for which (8) holds should be as large as possible. Nevertheless, this is the way to derive LR-acceptors (and presumably LL-acceptors as well).

It may be mentioned here that the construction of an acceptor is only one step in the construction of a parser, which has the additional tasks of building a derivation tree and of error detection or error recovery. Usually, the construction of a derivation tree can be implemented as a side-effect of the control flow of the acceptor. Error handling requires that the nontermination be replaced by the generation of adequate error messages. In the remainder of this paper we concentrate on the acceptance problem. Error detection is mentioned briefly in Section 8.

2. Notational conventions

In each section the formulae are numbered consecutively. For reference to formulae from other sections we use the convention that $i(j)$ refers to formula (j) of Section i.

Functions

The application of a function on an argument is denoted by the infix-operator ".". The operator "." binds stronger than all other operators. It binds from left to right, so as to allow currying.

Quantifications and sets

We write $(\forall x \in X: b.x: f.x)$ to denote the predicate that $f.x$ holds for all x, where the dummy x ranges through the elements of X that satisfy predicate b. If $b.x$ is

omitted, the default value *true* is meant. The indication "$\in X$" can be omitted if the type of x is clear from the context. Similar quantifications are

$(\exists x \in X: b.x: f.x)$: "$f.x$ holds for at least one x with $b.x$",

$(\bigcup x \in X: b.x: f.x)$: "the union of the $f.x$ with $b.x$",

$(\mathbf{SET}\ x \in X: b.x: f.x)$: "the set of the $f.x$ with $b.x$".

The empty set is denoted $\emptyset$. The braces "{" and "}" are only used to enclose comments within formal proofs.

Strings

Let A be a set of symbols. We use A^* to denote the set of the finite strings of elements of A. The elements of A are regarded as strings of length 1. So A is a subset of A^*. We write ε to denote the empty string. Catenation of strings is denoted by means of the infix operator ";" (by convention ";" and ε are not elements of A). The string functions *head* and *tail* are defined as usual: if $x = b;y$ with $b \in A$ then $head.x = b$ and $tail.x = y$. We also need the string function *last* which is given by

(0) $\quad last.(y;b) = b, \quad$ if $b \in A$.

For a string x the set of suffixes $Suff.x$ is given by

(1) $\quad y \in Suff.x \ \equiv\ (\exists z \in A^* :: x = z;y)$.

For any $y \in Suff.x$ we define string $x - y$ by

(2) $\quad x - y = z \ \equiv\ x = z;y$.

The length of a string x is denoted by $|x|$. If i is an integer with $0 \leqslant i \leqslant |x|$, we write $x|i$ to denote the prefix of x of length i. If X and Y are sets of strings, we write

(3) $\quad X;Y = (\mathbf{SET}\ x \in X, y \in Y :: x;y)$.

3. Two blind algorithms

We let $G = \langle V, T, P, S \rangle$ be a context-free grammar. Here, T is the finite set of the terminal symbols, V is the finite set of the nonterminal symbols, V and T are disjoint, $S \in V$ is the start symbol, and P is the set of productions. Formally speaking, P is a finite subset of the cartesian product $V \times A^*$ where A is the disjoint union of V and T.

We use the convention that the letters b, c, and d represent symbols in A and that the letters from m to z represent strings over A. In the programs we use strings s and z over A and strings m, p, q, r, and t over T; the letter m represents the constant input string, the other letters are program variables subject to the invariants $J0$ of (3) and $J1$ of (10) below.

Theory. In LR-parsing, we only consider rightmost derivations. These are formalised as follows. One step of a rightmost derivation is characterised by relation rm: for strings $x,y \in A^*$ we define

(0) $rm.x.y \equiv$

$$(\exists v \in A^*, w \in T^*, \langle c, z\rangle \in P :: x = v;c;w \wedge y = v;z;w).$$

The reflexive transitive closure of relation rm is denoted by rm^*. So, formally, we have

(1) $rm^*.x.y \equiv (\exists i: i \geq 0 : rm^i.x.y)$, where

$rm^0.x.y \equiv x = y$, and

$rm^{i+1}.x.y \equiv (\exists z \in A^* :: rm^i.x.z \wedge rm.z.y)$.

It is well known (cf. [6, Section 4.3]) that the language generated by grammar G is the set of the strings $m \in T^*$ with $rm^*.S.m$, that is

(2) $[m \in L.G \equiv m \in T^* \wedge rm^*.S.m]$.

Program. The task of the algorithm is to read a given string $m \in T^*$ and to decide whether $rm^*.S.m$ holds. We assume that the string is read from left to right. So we have the invariant relation $m = p;t$, where p has been read and t is the remainder of the input string m. Let string $s \in A^*$ represent the syntactic structure that has been recognised. Accordingly, we extend the invariant to

(3) $J0$: $m = p;t \wedge rm^*.s.p$.

This is known as the valid prefix property. Now the purpose of the algorithm is to establish (if possible) the validity of $rm^*.S.m$. We observe that

(4) $[J0 \wedge s = S \wedge t = \varepsilon \Rightarrow rm^*.S.m]$.

So we need two kinds of actions: to read string t and to process string s. The obvious candidate for a reading command is

(5) $C0 = |[\ s := (s;head.t)$

$\quad ;\ p := (p;head.t)$

$\quad ;\ t := tail.t\ ]|$.

We calculate the weakest precondition of $C0$ with postcondition $J0$ in

$wp.C0.J0$

$\Leftarrow$ {(5); (3); substitutions}

$t \neq \varepsilon \wedge m = p;head.t;tail.t \wedge rm^*.(s;head.t).(p;head.t)$

$\equiv$ {(0), (1) and $head.t \in T$}

$t \neq \varepsilon \wedge m = p;t \wedge rm^*.s.p$

$\equiv$ {(3)}

$J0 \wedge t \neq \varepsilon$.

This proves

(6) $[J0 \wedge t \neq \varepsilon \Rightarrow wp.C0.J0]$.

In processing string s we have more freedom. The second conjunct of (3) admits replacement of a segment z of s by a nonterminal c if $\langle c, z\rangle \in P$ and if z is followed in s only by terminal symbols. Given a derivation of m, we assume that such a reduction is performed as soon as is possible in the repetition. Therefore, we consider only reductions of suffixes z of s. So, we propose a command

(7) $D0 = |[\ s := ((s-z);c)\ ]|$.

An adequate precondition is found in

$wp.D0.J0$

$\equiv$ {(3), (7) and substitutions}

$m = p;t \wedge z \in Suff.s \wedge rm^*.((s-z);c).p$

$\Leftarrow$ {(0) and transitivity of rm^*}

$m = p;t \wedge z \in Suff.s \wedge \langle c, z\rangle \in P \wedge rm^*.s.p$

$\equiv$ {(3)}

$J0 \wedge \langle c, z\rangle \in P \wedge z \in Suff.s.$

This proves that

(8) $[J0 \wedge \langle c, z\rangle \in P \wedge z \in Suff.s \Rightarrow wp.D0.J0]$.

Adding the obvious initialisation we get the program

(9)
$P0 = |[\ p := \varepsilon;\ t := m;\ s := \varepsilon$
; **do** $\{J0\}\ t \neq \varepsilon \vee s \neq S \rightarrow$
if $t \neq \varepsilon \rightarrow C0$
[] **let** $\langle c, z\rangle \in P$ **with** $z \in Suff.s \rightarrow D0$
fi
od $\{J0 \wedge t = \varepsilon \wedge s = S$, and hence $rm^*.S.m\}$
$]|$.

One can easily verify the correctness of the initialisation. The body of the repetition is an alternative statement which terminates if and only if at least one of the guards holds. If the body terminates, it preserves the invariant $J0$ by (6) and (8). Even if the body always terminates, the loop need not terminate. When the loop terminates, $J0$ holds and $t = \varepsilon$ and $s = S$, so that $rm^*.S.m$ follows from (4). In that case, string m has been recognised (notice that $rm^*.S.m$ is constant). This shows that program $P0$ of (9) is conditionally correct, in the sense that formula 1(6) holds in the form

$[wap.P0.true \Rightarrow rm^*.S.m]$.

Actually, 1(7) holds as well, but the fact is not useful for the rest of the development (it follows from the result of Section 6 below).

The parsing program $P0$ of (9) may easily turn into a blind alley. In order to avoid this we shall allow more information concerning string t to appear in the guards of the alternative statement. More precisely, we fix a number $k \geq 0$ and we let the choice between a *shift* $C0$ or an adequate *reduction* $D0$ be guided by $(s;q)$ where q is the prefix of t of length k. To guarantee that t has a prefix of length k, we extend string m with a fixed string of length k, say $\perp^k$ where $\perp$ is a terminal symbol.

Remark. The decision to use a window of fixed size is rather arbitrary. The particular choice of the additional suffix $\perp^k$ is irrelevant for the derivation and correctness of our version of LR-parsing. For immediate error detection, however, it is useful to postulate that the first symbol of the suffix (here $\perp$ if $k > 0$) does not occur in the righthand side of any production. □

We now replace program variable t of $P0$ by the catenation of two program variables $q \in T^k$ and $r \in T^*$ with the invariant

(10) $\quad J1: \quad m;\perp^k = p;q;r \ \wedge \ rm^*.s.p.$

This invariant is established by the initialisation

(11) $\quad In1 = |[\ p := \varepsilon ; \ s := \varepsilon$

$\quad ; \ q := (m;\perp^k) \,|\, k \quad \{\text{the first } k \text{ symbols of } m;\perp^k\}$

$\quad ; \ r := tail^k.(m;\perp^k) \quad \{\text{the remainder}\} \ ]|.$

To preserve $J1$, command $C0$ is replaced by

(12) $\quad C1 = |[\ q := (q;head.r); \ r := tail.r$

$\quad ; \ s := (s;head.q); \ p := (p;head.q)$

$\quad ; \ q := tail.q \ ]|.$

A straightforward calculation (similar to the one that preceded (6)) yields

(13) $\quad [J1 \wedge r \neq \varepsilon \ \Rightarrow \ wp.C1.J1].$

Command $D0$ remains useful and we get

(14) $\quad [J1 \wedge \langle c, z\rangle \in P \wedge z \in Suff.s \ \Rightarrow \ wp.D0.J1].$

Now program $P0$ is replaced by

(15) $\quad P1 = |[\ In1$

$\quad ; \ \mathbf{do} \ \{J1\} \ r \neq \varepsilon \ \vee \ s \neq S \ \rightarrow$

$\quad\quad \mathbf{if} \ r \neq \varepsilon \ \rightarrow \ C1$

$\quad\quad [] \ \mathbf{let} \ \langle c, z\rangle \in P \ \mathbf{with} \ z \in Suff.s \ \rightarrow \ D0$

$\quad\quad \mathbf{fi}$

$\quad \mathbf{od} \ \{J1 \wedge r = \varepsilon \wedge s = S, \text{ and hence } rm^*.S.m\}$

$\quad]|.$

Now all remarks made after program $P0$ can be repeated. In fact, program $P1$ may have string q at its disposal, but it does not use this information.

4. A heuristic predicate

We now turn to the question of how to use the information contained in string $(s;q)$ for the guidance of the nondeterminacy of program $P1$ of 3(15). We strengthen the invariant so that choices that cannot be successful are recognised earlier. To this end, we introduce a condition that expresses that $(s;q)$ is a prefix of a rightmost derivation of $(S;\perp^k)$, and that string $s - last.s$ is fully interpreted, in the sense that none of its substrings will be replaced by a nonterminal (compare $D0$ in 3(7)).

In other words, the condition is that $(s;q)$ can be obtained from $(S;\perp^k)$ by zero or more steps of the following type. If $last.s$ is a terminal symbol, the last symbol of $(s;q)$ is removed. If $last.s$ is a nonterminal, it is rewritten. Nonterminals in $s - last.s$ are not rewritten since that would yield a string s' in which $s' - last.s'$ is not fully interpreted.

Theory. The set of strings that end in at least k terminal symbols is denoted $(A^*;T^k)$ (cf. 2(3)). Now the steps suggested above are formalised in relation rk on $(A^*;T^k)$, defined by

$$(0) \qquad rk.x.y \;\equiv\; (\exists b \in T :: x = y;b) \;\vee\; (\exists v \in A^*, w \in T^k, \langle c, z\rangle \in P :: x = v;c;w \;\wedge\; y = v;z;w).$$

Notice that, since $y \in (A^*;T^k)$, the first disjunct implies that the last $k+1$ symbols of string x are terminals. We define relation rk^* on $(A^*;T^k)$ to be the reflexive transitive closure of rk. Its formal definition is analogous to 3(1). Predicate $rk^*.x.y$ holds if and only if there is a sequence of strings from x to y in which consecutive pairs satisfy rk. Such a sequence is called an rk-derivation.

The name rk stands for rightmost prefix with respect to number k. The condition is a variation of definition 3(0), but the trailing string of terminal symbols is forced onto a bed of Procrustes of length k.

Program. We extend invariant $J1$ of 3(10) with the "heuristic" predicate

$$(1) \qquad H: \quad rk^*.(S;\perp^k).(s;q).$$

Predicate H is only introduced to guide the nondeterminacy. It follows from formula 5(1) below that predicate H implies the existence of a string $u \in T^*$ with $rm^*.(S;\perp^k).(s;q;u)$, so that the execution is not yet in a blind alley.

Remark. Instead of the choices (0) and (1), one can propose to use the relation rp given by

$$rp.x.y \;\equiv\; (\exists u \in T^* :: rm^*.x.(y;u))$$

and the heuristic predicate $rp.(S;\perp^k).(s;q)$. The formulae (2), (3), and (4) below can be adapted immediately. Lemma (8) below, however, has an analogue for strings $w \in (V;T^k)$ but not for strings $w \in T^{k+1}$ and we need both cases. □

We start with an investigation of the preservation of $J1 \wedge H$ under the commands $C1$ and $D0$. It is clear from (1), 3(12), and 3(7) that

(2) $\quad [wp.C1.(J1 \wedge H) \equiv wp.C1.J1 \wedge rk^*.(S;\perp^k).(s;q;head.r)],$

(3) $\quad [wp.D0.(J1 \wedge H) = wp.D0.J1 \wedge rk^*.(S;\perp^k).((s-z);c;q)].$

The suffixes $(q;head.r)$ and $(c;q)$ are both elements of $(A;T^k)$, i.e. they have length $k+1$ and have a suffix of k terminals. This observation suggests to make an analysis of $rk^*.(S;\perp^k).(x;w)$ with $x \in A^*$ and $w \in (A;T^k)$.

Theory. For a string $x \in A^*$, we define the subset $N.x$ of $(A;T^k)$ by

(4) $\quad w \in N.x \equiv rk^*.(S;\perp^k).(x;w).$

In view of (2) and (3), we shall need $N.x$ for $x = s$ and for prefixes x of s. Now the aim is to derive a recurrence equation for function N. For $x \in A^*$ and $w \in (A;T^k)$ we observe

$$
\begin{aligned}
& w \in N.x \wedge x \neq \varepsilon \\
\equiv \quad & \{(4)\} \\
& rk^*.(S;\perp^k).(x;w) \wedge x \neq \varepsilon \\
\equiv \quad & \{\text{use (8) below with } n := (S;\perp^k)\} \\
& (\exists v, y, z \in A^*, c \in V, t \in T^k : x = v;y \wedge y \neq \varepsilon \wedge \langle c,y;z\rangle \in P : \\
& rk^*.(S;\perp^k).(v;c;t) \wedge rk^*.(z;t).w) \\
\equiv \quad & \{(4);\ 2(1), \text{ and } 2(2)\} \\
& (\exists y \in Suff.x, z \in A^*, c \in V, t \in T^k : y \neq \varepsilon \wedge \langle c,y;z\rangle \in P : \\
& (c;t) \in N.(x-y) \wedge rk^*.(z;t).w).
\end{aligned}
$$

For any string $x \in (A^*;T^k)$, we now define $Rks.x$ as the subset of $(A;T^k)$ given by

(5) $\quad w \in Rks.x \equiv rk^*.x.w.$

Now the above calculation implies that, for string $x \neq \varepsilon$,

(6) $\quad N.x =$

$$
\begin{aligned}
& (\bigcup y \in Suff.x, z \in A^*, c \in V, t \in T^k : \\
& \quad y \neq \varepsilon \wedge \langle c, y;z\rangle \in P \wedge (c;t) \in N.(x-y) : Rks.(z;t)).
\end{aligned}
$$

On the other hand, it follows from (4) and (5) that

(7) $\quad N.\varepsilon = Rks.(S;\perp^k).$

The sets $Rks.(z;t)$ need only be computed for finitely many strings: in fact, z ranges over the finite set of suffixes of productions and t ranges over the finite set T^k. Function Rks can be effectively computed. Since it only depends on the grammar

(not on the input string), we may assume that it is known after inspection of the grammar. This issue is postponed to Section 9. There it turns out that the usual delicacies of nullable nonterminals are encapsulated in *Rks*. Now the equations (6) and (7) can effectively be used to compute $N.x$ for any string x.

It remains to prove the technical result used above.

Lemma. *For* $x \in A^*$ *and* $n, w \in (A^*;T^k)$ *with* $|n| \leq k+1 \leq |w|$ *we have*

(8) $rk^*.n.(x;w) \wedge x \neq \varepsilon \equiv$

$(\exists v, y, z \in A^*, c \in V, t \in T^k : x = v;y \wedge y \neq \varepsilon \wedge \langle c, y;z \rangle \in P:$

$rk^*.n.(v;c;t) \wedge rk^*.(z;t).w).$

Proof. Let the lefthand side of (8) be given. Choose an rk-derivation from n to $(x;w)$. In this derivation, choose the first derivative of n of the form $(x;p)$ with $p \in (A^*;T^k)$ and $rk^*.p.w$. Since $|w| \geq k+1$, we have $|p| \geq k+1$. Since $x \neq \varepsilon$, it follows that $|x;p| > k+1$ and hence $n \neq (x;p)$. Let u be the predecessor of $(x;p)$ in the derivation. Since $(x;p)$ is the first occurrence, we have $u \neq (x;p;b)$ for all $b \in T$. Therefore, from (0), we have $u = (v;c;t)$ with $v \in A^*$, $c \in V$, and $t \in T^k$ and there is an $s \in A^*$ with $\langle c, s \rangle \in P$ and $v;s;t = x;p$. Again using that $(x;p)$ is the first occurrence, we see that x is not a prefix of v. It follows that

$$|s;t| > |p| > k = |t|,$$

so that we can split s into parts y and z with

$$s = y;z \wedge y \neq \varepsilon \wedge v;y = x \wedge z;t = p.$$

Thus we obtain the righthand side of (8).

Conversely, let the righthand side of (8) be given. Since $x = v;y$ and $y \neq \varepsilon$, we have $x \neq \varepsilon$. On the other hand, by (0), the data imply $rk.(v;c;t).(x;z;t)$. Since $rk^*.(z;t).w$, we have $rk^*.(x;z;t).(x;w)$. By transitivity it follows that $rk^*.n.(x;w)$. This proves the lefthand side of (8). □

Program. After these preparations we come back to the algorithm. With respect to command $C1$ we observe

(9) $wp.C1.(J1 \wedge H)$

$\equiv$ {(2) and (4)}

$wp.C1.J1 \wedge q;head.r \in N.s$

$\Leftarrow$ {3(13)}

$J1 \wedge r \neq \varepsilon \wedge q;head.r \in N.s.$

With respect to command $D0$ we observe

$$
\begin{array}{ll}
(10) & wp.D0.(J1 \wedge H) \\
\equiv & \{(3) \text{ and } (4)\} \\
& wp.D0.J1 \wedge c;q \in N.(s-z) \\
\Leftarrow & \{3(14)\} \\
& J1 \wedge \langle c, z\rangle \in P \wedge z \in Suff.s \wedge c;q \in N.(s-z).
\end{array}
$$

In the preconditions given by (9) and (10), predicate $J1$ occurs instead of $J1 \wedge H$. Since the initialisation $s = \varepsilon$ and the postcondition $s = S$ are incompatible, the body of the repetition is called at least once. Therefore, we may use the body to establish $J1 \wedge H$ (if possible). This proves conditional correctness of the annotated program

$$
\begin{array}{ll}
(11) & P2 = |[\ In1 \\
& \quad ;\ \textbf{do}\ \{J1\}\ r \neq \varepsilon \vee s \neq S \rightarrow \\
& \qquad \textbf{if}\ r \neq \varepsilon \wedge q;head.r \in N.s \rightarrow C1 \\
& \qquad [] \ \textbf{let}\ \langle c, z\rangle \in P \\
& \qquad\quad \textbf{with}\ z \in Suff.s \wedge c;q \in N.(s-z) \rightarrow D0 \\
& \qquad \textbf{fi}\ \{J1 \wedge H\} \\
& \quad \textbf{od}\ \{J1 \wedge r = \varepsilon \wedge s = S, \text{ and hence } rm^*.S.m\} \\
&]|.
\end{array}
$$

That is, program $P2$ satisfies condition 1(6) in the form

$$(12) \qquad [wap.P2.true \Rightarrow rm^*.S.m].$$

In Section 6 below, we prove that the implication can be replaced by an equivalence. This means that program $P2$ is an angelic acceptor of the language $L.G$, see 1(7). It follows from the result of Section 8 that, if grammar G is LR(k), program $P2$ is deterministic and, hence, a genuine acceptor. Therefore, program $P2$ can be our final result.

In program $P2$, we need the values of $N.x$ for some prefixes x of s. The sets $N.x$ can be computed by means of (6) and (7). Since string s is only modified at its tail, it is more efficient to introduce a stack Nst of subsets of $(A;T^k)$, say with stack pointer h and invariant

$$Jn: \quad h = |s| \wedge (\forall i: 0 \leq i \leq h: Nst.i = N.(s\,|\,i)).$$

In order to make Jn an invariant of the repetition of $P2$, it suffices to extend each of the commands $In1$, $C1$ and $D0$ with the restoration command

$$
\begin{array}{l}
|[\ h := |s| \quad \{(\forall i: 0 \leq i < h: Nst.i = N.(s\,|\,i))\} \\
\quad ;\ Nst.h := N.s \quad \{\text{use (6) and (7)}\}\ ]|.
\end{array}
$$

Here, $Nst.h$ can be computed in constant time from its predecessors (for a fixed grammar). The details are left to the reader. In Section 7 below, we present the usual LR(k) algorithm with its finite automaton and its stack of item sets as another optimisation of program $P2$.

5. The relevance of *rk*-derivations

Up to this point we did not use any relationship between relation rm^* of Section 3 and relation rk^* of Section 4. The fact that every string of the language can be accepted by program $P2$, however, depends on the connections between the two relations. There are two kinds of connections: rk^* implies some form of rm^* and rm^* implies some form of rk^*.

Theory. The first connection is that for $x, y \in (A^*; T^k)$ we have

(0) $\quad rk^*.x.y \Rightarrow (\exists u \in T^* :: rm^*.x.(y;u))$.

So, $rk^*.x.y$ implies that y is a prefix of a rightmost derivative of x.

Formula (0) is proved by induction on the length of the rk-derivation. The base case is $x = y$, in which case the consequent of (0) holds with $u = \varepsilon$. The induction step is

$$
\begin{aligned}
& rk^*.x.y \wedge x \neq y \\
\Rightarrow\ & \{\text{one step and the induction hypothesis (0) with } x := z\} \\
& (\exists z \in (A^*; T^k), u \in T^* :: rk.x.z \wedge rm^*.z.(y;u)) \\
\equiv\ & \{4(0)\} \\
& (\exists z \in (A^*; T^k), u \in T^* :: \\
& \quad ((\exists b \in T :: x = z;b) \\
& \quad \vee (\exists v \in A^*, t \in T^k, \langle c, w\rangle \in P :: x = v;c;t \wedge z = v;w;t)) \\
& \quad \wedge rm^*.z.(y;u)) \\
\Rightarrow\ & \{3(0)\} \\
& (\exists z \in (A^*; T^k), u \in T^* :: ((\exists b \in T :: x = z;b) \vee rm.x.z) \wedge rm^*.z.(y;u)) \\
\Rightarrow\ & \{\text{calculus, 3(0) and 3(1)}\} \\
& (\exists u \in T^* :: (\exists b \in T :: rm^*.x.(y;u;b)) \vee rm^*.x.(y;u)) \\
\equiv\ & \{\text{calculus}\} \\
& (\exists u \in T^* :: rm^*.x.(y;u)).
\end{aligned}
$$

The converse connection is more complicated. It is expressed in

(1) **Theorem.** *Let* $x \in (A^*; T^k)$, $y \in A^*$, *and* $t \in T^*$ *be such that*

(2) $\quad rm^*.x.(y;t) \wedge (y = \varepsilon \vee last.y \in V)$.

Then $|t| \geq k$ *and* $rk^*.x.(y;(t|k))$.

Proof. The proof uses induction on the length of the *rm*-derivation from x to $(y;t)$. If the derivation has length 0, then $x=(y;t)$ and hence $y;t\in(A^*;T^k)$, so that $|t|\geq k$ by the second conjunct of (2), and that $rk^*.x.(y;(t|k))$ follows from repeated application of the first disjunct of 4(0).

In the case that the derivation is nontrivial, we may split off the last step of the *rm*-derivation. By 3(0), there are $u\in A^*$, $c\in V$, $v\in T^*$, and $\langle c, z\rangle\in P$ with

(3) $\quad rm^*.x.(u;c;v)\ \wedge\ u;z;v=y;t.$

By the induction hypothesis with $y:=(u;c)$ and $t:=v$, it follows that $|v|\geq k$ and

$$rk^*.x.(u;c;(v|k)).$$

Since $\langle c, z\rangle\in P$, it follows with 4(0) that

(4) $\quad rk^*.x.(u;z;(v|k)).$

Since $v\in T^*$, the second conjuncts of (2) and (3) imply that y is a prefix of $(u;z)$, and hence that v is a suffix of t. This implies that $|t|\geq k$ and that

$$u;z;(v|k)=y;(t|k);w \quad \text{for some } w\in T^*.$$

Repeated application of the first disjunct of 4(0) now implies

(5) $\quad rk^*.(u;z;(v|k)).(y;(t|k)).$

By transitivity, $rk^*.x.(y;(t|k))$ follows from (4) and (5). □

We need the following two applications of Theorem (1).

(6) **Corollary.** *Let $b\in A$ and $t\in T^*$ with $rm^*.b.t$. Then*

$$(\forall u\in T^k::rk^*.(b;u).((t;u)|k)).$$

Proof. We have $rm^*.(b;u).(t;u)$. So the assertion follows from Theorem (1) with $x:=b;u$ and $y:=\varepsilon$ and $t:=t;u$. □

(7) **Corollary.** *Let $v\in A^*$, $t\in T^k$, and $c\in V$ with*

$$rm^*.(S;\perp^k).(v;c;t).$$

(a) *Then $|t|\geq k$ and $(c;(t|k))\in N.v$.*
(b) *For all $s\in A^*$, $u\in T^*$, and $\langle c, z\rangle\in P$ we have*

$$s;u=v;z;t\ \wedge\ |s|<|v;z|\ \Rightarrow\ |u|>k\wedge(u|k+1)\in N.s.$$

Proof. (a) It follows from Theorem (1) with $x:=(S;\perp^k)$ and $y:=v;c$ that $|t|\geq k$ and

(8) $\quad rk^*.(S;\perp^k).(v;c;(t|k)),$

so that $(c;(t|k))\in N.v$ by definition 4(4).

(b) Since $|s|<|v;z|$, we have $|u|>|t|\geq k$. The second assertion is proved in

$(u\,|\,k+1)\in N.s$

$\equiv$ {definition 4(4)}

$rk^*.(S;\perp^k).(s;(u\,|\,k+1))$

$\Leftarrow$ {$s;(u\,|\,k+1)$ is a prefix of $(v;z;(t\,|\,k))$, $u\in T^*$, and 4(0)}

$rk^*.(S;\perp^k).(v;z;(t\,|\,k))$

$\equiv$ {4(0) and $\langle c, z\rangle\in P$ and (8)}

true. □

6. Angelic guidance suffices

Program. We claim that under angelic nondeterminacy any string m in the language of the given grammar can be recognised by program $P2$ of 4(11). Since recognition is equivalent to termination of $P2$, this means that program $P2$ may terminate for the given string m. Formally speaking, the claim is

(0) $\quad [rm^*.S.m \Rightarrow wap.P2.true]$,

which is the converse of implication 4(12). This section is devoted to the proof of formula (0).

In order to prove (0), we assume $rm.^*S.m$. Then we can choose a rightmost derivation of string m. Let $x.i$ with $0\leq i\leq g$ be the consecutive sentential forms of the derivation, so that

(1) $\quad x.0=S \wedge x.g=m \wedge (\forall i: 0<i\leq g: rm.(x.(i-1)).(x.i))$.

Notice that this implies $rm^*.(S;\perp^k).(x.i;\perp^k)$ for $0\leq i\leq g$.

It is clear that we may assume

(2) $\quad (\forall i: 0<i\leq g: x.i\neq S)$.

By 3(0), it follows from (1) that, for $0<i\leq g$, there exist productions $\langle c.i, z.i\rangle\in P$ and decompositions

(3) $\quad x.(i-1)=v.i;c.i;w.i,$

$\quad x.i=v.i;z.i;w.i,$

with $v.i\in A^*$ and $w.i\in T^*$.

We force program 4(11) to accept string $m=x.g$ by strengthening the guards of the alternative statement in the body of the repetition. For this purpose we introduce a ghost variable i of type integer with the additional invariant

(4) $\quad K: \quad 0\leq i\leq g \wedge s;q;r=x.i;\perp^k \wedge (i=0\vee|s|\leq|v.i;z.i|)$.

The third conjunct is motivated by Corollary 5(7). It was announced in the first paragraph of Section 4 by saying that no substring of $s - last.s$ needs to be replaced by a nonterminal. The conventional expression is that s is a viable prefix (cf. [6, p. 249]).

It is easy to see that predicate K is initialised by the extended initialisation

(5) $In3 - |[\ In1$ {cf. 3(11)}
$\quad ;\ i := g\ ]|.$

Since $x.0 = S \in V$ and $q \in T^k$ and $r \in T^*$, it follows from (1) and (2) that

(6) $[K \Rightarrow (i = 0 \equiv r = \varepsilon \wedge s = S)].$

This proves that the guard of the repetition of 4(11) may be replaced by $i \neq 0$.

We extend commands $C1$ and $D0$ in order to keep predicate K invariant. Actually, $C1$ need not be changed. Since we want to strengthen the guard of $C1$ in 4(11), we observe

(7) $r \neq \varepsilon \wedge q;head.r \in N.s \wedge wp.C1.(J1 \wedge H \wedge K)$

$\Leftarrow$ {4(9)}

$J1 \wedge r \neq \varepsilon \wedge q;head.r \in N.s \wedge wp.C1.K$

$\equiv$ {(4) and definition $C1$ in 3(12); calculus}

$J1 \wedge r \neq \varepsilon \wedge q;head.r \in N.s \wedge 0 \leq i \leq g$

$\wedge\ s;q;r = x.i;\perp^k \wedge (i = 0 \vee |s| < |v.i;z.i|)$

$\equiv$ {(4); (6) gives $i \neq 0$;

$|s| < |v.i;z.i|$ implies $|q;r| > k$ so that $r \neq \varepsilon$}

$J1 \wedge K \wedge i \neq 0 \wedge q;head.r \in N.s \wedge |s| < |v.i;z.i|$

$\equiv$ {Corollary 5(7)(b) with $u := q;r$}

$J1 \wedge K \wedge i \neq 0 \wedge |s| < |v.i;z.i|.$

This suggests to replace the first alternative in 4(11) by

$|s| < |v.i;z.i| \rightarrow C1.$

In view of the postulated invariance of K, it remains to produce an alternative with guard $s = v.i;z.i$. We now expect an application of $D0$ of 3(7). Production $\langle c.i, z.i \rangle$ is an obvious candidate for usage in $D0$. Since the usage of a production is a step in the derivation, we also need $i := i - 1$. So, since $s - z.i = v.i$, we propose the refinement $D3$ of $D0$ given by

(8) $D3 = |[\ s := v.i;c.i$
$\quad ;\ i := i - 1\ ]|.$

Since we want to strengthen the guard of $D0$ in 4(11) we observe that, under assumption of $0 < i \leq g$ and $s = v.i;z.i$,

(9) $c.i;q \in N.(v.i) \wedge wp.D3.(J1 \wedge H \wedge K)$

$\Leftarrow$ {4(10)}

$J1 \wedge c.i;q \in N.(v.i) \wedge wp.D3.K$

$\Leftarrow$ {(3), (4), and Corollary 5(7)(a)}

$J1 \wedge K \wedge wp.D3.K$

$\equiv$ {definition of K in (4) and of $D3$ in (8); calculus}

$J1 \wedge K \wedge v.i;c.i;q;r = x.(i-1);\perp^k$

$\wedge\ (i-1 = 0 \vee |v.i;c.i| \leq |v.(i-1);z.(i-1)|)$

$\equiv$ {remaining proof obligation, see (10) below}

$J1 \wedge K.$

It remains to observe that, for $0 < i \leq g$,

(10) $v.i;c.i;q;r = x.(i-1);\perp^k$

$\wedge\ (i-1 = 0 \vee |v.i;c.i| \leq |v.(i-1);z.(i-1)|)$

$\equiv$ {(3) and $c.i \in V$ and $w.(i-1) \in T^*$}

$v.i;z.i;q;r = x.i;\perp^k$

$\Leftarrow$ {(4)}

$K \wedge s = v.i;z.i.$

Putting (7) and (9) together we have proved conditional correctness of the annotated program $P3$ given by

(11)
```
P3 = |[ In3   {cf. (5)}
     ; do {J1 ∧ K} i ≠ 0
         if |s| < |v.i;z.i| →
           {r ≠ ε ∧ q;head.r ∈ N.s} C1
         [] s = v.i;z.i →
           {c.i;q ∈ N.(v.i)} D3
         fi
       od {J1 ∧ K ∧ i = 0, and hence rm*.S.m}
     ]|.
```

Since K implies the disjunction of the guards of the alternative statement, the body of the loop terminates. The loop itself also terminates, as is easily seen, since the variant function $i + |r|$ decreases with 1 in each of the commands $C1$ and $D3$. Therefore, program $P3$ terminates. The assertions in $P3$ show that $P3$ is a refinement of program $P2$ of 4(11). This proves that $P2$ can be guided towards termination, i.e. that $wap.P2.true$ holds for the given string m. This concludes the proof of (0).

7. The birth of the automaton

The recurrence equations for $N.x$ given in 4(6) and 4(7) may admit (for a fixed grammar) computation in constant time, but are yet inefficient, and inadequate for preprocessing. The same holds for the pattern matching required in the second alternative of program 4(11). In this section we describe the standard LR(k)-parsing algorithm as an optimisation of program $P2$ in which these inefficiencies are avoided.

The idea is that both computations mentioned above require knowledge of the strings $c;t \in N.(x-y)$ where y is a suffix of x that allows a production $\langle c, y;z\rangle$. This suggests the three definitions of the next paragraph.

Theory. An *item* is defined to be a quadruple $\langle c, y, z, t\rangle$ with $c \in V$, $y, z \in A^*$, $t \in T^k$, and $\langle c, y;z\rangle \in P$. We write *Item* to denote the set of all items. Notice that *Item* is a finite set because of the finiteness of T^k and P. For a string $x \in A^*$ we define $M.x$ as the subset of *Item* given by

(0) $\quad \langle c, y, z, t\rangle \in M.x \;\equiv\; y \in Suff.x \;\wedge\; c;t \in N.(x-y).$

Now formula 4(6) implies that, for $x \neq \varepsilon$,

(1) $\quad N.x = (\bigcup \langle c, y, z, t\rangle \in M.x: y \neq \varepsilon: Rks.(z;t)).$

We want to eliminate function N from the program. Recall from 4(4) that $N.x$ is a subset of $(A;T^k)$. For every $x \in A^*$ we define

(2) $\quad E.x = T^{k+1} \cap N.x.$

Program. Because of definitions (0) and (2), and the fact that $q \in T^k$, program $P2$ of 4(11) is equivalent to

(3)
```
|[ In1
; do {J1} r ≠ ε ∨ s ≠ S →
      if r ≠ ε ∧ q;head.r ∈ E.s → C1
      [] let c, z with ⟨c, z, ε, q⟩ ∈ M.s → D0
      fi {J1 ∧ H}
  od {J1 ∧ r = ε ∧ s = S, and hence rm*.S.p}
]|.
```

It remains to compute the sets $E.s$ and $M.s$.

Theory. The set $M.\varepsilon$ is determined by observing that, for every item $\langle c, y, z, t\rangle$,

$\langle c, y, z, t\rangle \in M.\varepsilon$

$\equiv$ {(0)}

$y = \varepsilon \;\wedge\; c;t \in N.\varepsilon$

$\equiv$ {4(7)}

$y = \varepsilon \;\wedge\; c;t \in Rks.(S;\perp^k).$

This implies that

(4) $M.\varepsilon = (\text{SET } c, z, t: \langle c, z\rangle \in P \wedge c;t \in Rks.(S;\perp^k): \langle c, \varepsilon, z, t\rangle).$

We determine a recurrence equation for $M.x$, in which $M.(x;b)$ is expressed in terms of $M.x$. For every item $\langle c, y, z, t\rangle$ with $y \neq \varepsilon$ we observe that

(5) $\langle c, y, z, t\rangle \in M.(x;b)$

$\equiv$ {(0); we give ";" higher priority than "−"}

$y \in Suff.(x;b) \wedge c;t \in N.(x;b-y)$

$\equiv$ {$y \neq \varepsilon$ and calculus}

$b \in Suff.y \wedge y-b \in Suff.x \wedge c;t \in N.(x-(y-b))$

$\equiv$ {(0)}

$b \in Suff.y \wedge \langle c, y-b, (b;z), t\rangle \in M.x.$

For items of the form $\langle c, \varepsilon, z, t\rangle$ we have

(6) $\langle c, \varepsilon, z, t\rangle \in M.(x;b)$

$\equiv$ {(0)}

$c;t \in N.(x;b)$

$\equiv$ {(1) and $x;b \neq \varepsilon$}

$(\exists \langle d, w, v, q\rangle \in M.(x;b): w \neq \varepsilon: c;t \in Rks.(v;q))$

$\equiv$ {(5) with $w \neq \varepsilon$; introduce $u = w-b$}

$(\exists d, u, v, q: \langle d, u, (b;v), q\rangle \in M.x: c;t \in Rks.(v;q)).$

Traditionally, the items of (5) are called kernel items and the items of (6) are called closure items.

Combining (5) and (6), we obtain for every item $\langle c, y, z, t\rangle$

(7) $\langle c, y, z, t\rangle \in M.(x;b) \equiv$

$(\exists d, u, v, q: \langle d, u, (b;v), q\rangle \in M.x:$

$\langle c, y, z, t\rangle = \langle d, (u;b), v, q\rangle \vee (y = \varepsilon \wedge c;t \in Rks.(v;q))).$

Since *Item* is a finite set, formula (7) suggests the following form of preprocessing. For any subset U of *Item* and any symbol $b \in A$, we define $step.b.U$ as the subset of *Item* given by

(8) $\langle c, y, z, t\rangle \in step.b.U \equiv$

$(\exists d, u, v, q: \langle d, u, (b;v), q\rangle \in U:$

$\langle c, y, z, t\rangle = \langle d, (u;b), v, q\rangle \vee (y = \varepsilon \wedge c;t \in Rks.(v;q))).$

Then formula (7) reduces to

(9) $M.(x;b) = step.b.(M.x).$

Let Q be the power set of the finite set *Item*. Now *step* is a function $A \to (Q \to Q)$. The triple $\langle Q, step, M.\varepsilon \rangle$ is a version of the finite automaton that usually occurs in LR-parsing. Function *step* can be computed in the preprocessing phase, for it clearly does not depend on the input string.

We now want to eliminate function E of formula (2) from the program. It follows from (2) and 4(7) that

(10) $\quad E.\varepsilon = T^{k+1} \cap Rks.(S;\perp^k).$

It follows from (1) and (2) that, for $x \neq \varepsilon$,

(11) $\quad E.x = e.(M.x)$

where, for every subset U of *Item*, the set $e.U$ is defined by

(12) $\quad e.U = (\bigcup \langle c, y, z, t \rangle \in U: y \neq \varepsilon: T^{k+1} \cap Rks.(z;t)).$

Clearly, function e is a function from Q to the power set of T^{k+1} that can be computed in the preprocessing phase.

Program. We now come back to the algorithm. In the program of (3) we need the values of $E.s$ and $M.s$ for the application of the two commands. Therefore, after the application we need the values of $E.s'$ and $M.s'$ where s' is the new value of s. Since s' is always nonempty, we can use formula (11) to compute $E.s'$ from $M.s'$.

In the case of $C1$, we have $s' = (s;b)$ with $b \in T$, so that $M.s'$ can be obtained from $M.s$ with formula (9). In the case of $D0$ we have $s' = ((s-z);c)$ with $\langle c, z \rangle \in P$, so that $M.s'$ can be obtained from $M.(s-z)$. Therefore, we introduce a stack *Mst* for the values $M.x$ where x is a prefix of s. Let h be the stack pointer (the identifiers s and p being in use for strings). We introduce a variable F to keep the value of $E.s$. The invariant $J1$ is extended accordingly with

(13) $\quad Jm: \quad F = E.s \wedge h = |s| \wedge (\forall i: 0 \leq i \leq h: Mst.i = M.(s \mid i)).$

It is clear that *Jm* is initialised by

(14) $\quad In4 = |[\ In1 \quad \{\text{cf. } 3(11)\}$
$\quad\quad ;\ h := 0$
$\quad\quad ;\ F := E.\varepsilon \quad \{\text{cf. } (10)\}$
$\quad\quad ;\ Mst.0 := M.\varepsilon \quad \{\text{cf. } (4)\}\ ]|.$

To preserve *Jm*, command $C1$ is extended to

(15) $\quad C4 = |[\ Mst.(h+1) := step.(head.(q;r)).(Mst.h)$
$\quad\quad ;\ h := h+1$
$\quad\quad ;\ F := e.(Mst.h)$
$\quad\quad ;\ C1\ ]|.$

Instead of $head.(q;r)$ we may write $head.q$ if $k > 0$ and $head.r$ if $k = 0$.

Command $D0$ is extended to

(16) $D4 = |[\ Mst.(h-|z|+1) := step.c.(Mst.(h-|z|))$

$;\ h := h-|z|+1$

$;\ F := e.(Mst.h)$

$;\ s := ((s-z);c)\ ]|.$

Now the program of (3) is equivalent to our final result

(17) $P4 = |[\ In4$

$;$ **do** $\{J1 \wedge Jm\}\ r \neq \varepsilon \vee s \neq S \rightarrow$

if $r \neq \varepsilon \wedge q;head.r \in F \rightarrow C4$

$[]$ **let** c, z **with** $\langle c, z, \varepsilon, q\rangle \in Mst.h \rightarrow D4$

fi $\{J1 \wedge Jm \wedge H\}$

od $\{J1 \wedge r = \varepsilon \wedge s = S$, and hence $rm^*.S.m\}$

$]|.$

Since program $P4$ is equivalent to $P2$, it follows from 4(12) and 6(0) that

(18) $[rm^*.S.m \equiv wap.P4.true]$,

which is the analogue of formula 1(7).

8. The determinstic acceptor

Program. As argued in Section 1, formula 7(18) implies that program $P4$ is an acceptor of the language if it is deterministic (cf. 1(8)). The nondeterminacy of $P4$ consists of the occurrence of two points of choice. One point is the choice between the first and the second alternative. If this choice occurs, it is called a shift-reduce conflict. The other point is the choice of a production $\langle c, z\rangle$ in the second alternative. If this choice occurs, it is called a reduce-reduce conflict.

More formally, let us consider triples (q, F, U) with $q \in T^k$, $F \subset T^{k+1}$, and $U \subset Item$. We define a triple (q, F, U) or the pair (q, F) to *allow a shift* if and only if $q;b \in F$ for some $b \in T$. We define the triple (q, F, U) or the pair (q, U) to *allow a reduction* $\langle c, z\rangle \in P$ if and only if $\langle c, z, \varepsilon, q\rangle \in U$. A triple is defined to *allow conflicts* if and only if it allows a shift and a reduction or at least two different reductions. If all reachable triples do not allow conflicts, grammar G is said to be LR(k) (cf. [7]).

Let grammar G be LR(k). This implies that a string m belongs to $L.G$ if and only if $P4$ terminates. So, if $m \notin L.G$, the program does not terminate. One possibility is that the alternative statement aborts since $(q, Mst.h)$ does not allow a reduction and either $r = \varepsilon$ or $(q;head.r) \notin F$. In this case, abortion can be replaced by an error message. In principle, another possibility would be that the repetition of $P4$ does not terminate, but we now show that this is not possible.

(0) **Theorem.** *Let G be* LR(k) *and $m \notin L.G$. Then the alternative statement in the body of the repetition of P4 aborts in a state where $(q,Mst.h)$ does not allow a reduction and either $r = \varepsilon$ or $(q;head.r) \notin F$.*

Proof. Since $P4$ is equivalent to $P2$, we may use $P2$ to argue about. The body of the repetition of $P2$ establishes $J1 \wedge H$. Now we observe

$$
\begin{aligned}
& J1 \wedge H \\
\equiv\ & \{4(1)\} \\
& J1 \wedge rk^*.(S;\perp^k).(s;q) \\
\Rightarrow\ & \{\text{formula } 5(0)\} \\
& J1 \wedge (\exists u \in T^* :: rm^*.(S;\perp^k).(s;q;u)) \\
\Rightarrow\ & \{3(10), 3(0), \text{and calculus}\} \\
& m;\perp^k = p;q;r \wedge (\exists u \in T^* :: rm^*.(S;\perp^k).(p;q;u)).
\end{aligned}
$$

Since $m \notin L.G$, we have $\neg rm^*.(S;\perp^k).(m;\perp^k)$. If string r would be equal to string u of the existential quantification, then program $P2$ can terminate. Since grammar G is LR(k), program $P2$ is deterministic. So, if $r = u$, program $P2$ necessarily terminates. Therefore, the number of steps performed without inspection of r is finite. This implies that the number of reductions performed in states with fixed p and q is finite. Since input string m is finite the number of shifts is finite. This implies that the only source of nontermination is the alternative statement. □

9. The determination of function *Rks*

Theory. In this section we finally obtain an effective characterisation of the crucial function *Rks* introduced in 4(5), and used in 7(6) and 7(8). It is here that the possibility of ε-productions complicates the algorithm and the proof. It turns out that our definition of *Rks* has served to isolate this complication, which in Knuth's paper led to the introduction of the function $H'_k(\sigma)$ (cf. [7], p. 615]).

Recall from 4(5) that, for $x \in (A^*;T^k)$, the set $Rks.x$ is the subset of $(A;T^k)$ given by

(0) $\quad w \in Rks.x \ \equiv\ rk^*.x.w.$

So, the determination of *Rks* depends on the determination of $rk^*.x.w$ for $w \in (A;T^k)$.

Since an rk-derivation from x to w may pass through strings of arbitrary length, it is not obvious that $rk^*.x.w$ can effectively be determined. We shall enforce finiteness of computation by transforming the derivations in such a way that the intermediate strings have bounded length. We first extend the freedom of relation rk drastically

by introducing a relation *frk* (for free *rk*). In the second step, the freedom of *frk* is reduced again by introducing a relation *brk* (for bounded *rk*) in which productive rewritings are only allowed in small strings.

The transformation requires special treatment of the *nullable* nonterminals, i.e. the elements of the set $V0$ defined by

(1) $\quad c \in V0 \;\equiv\; c \in V \wedge rm^*.c.\varepsilon,$

and of the *productive* symbols: the elements of the set $A1$ defined by

(2) $\quad b \in A1 \;\equiv\; (\exists u \in T^* :: rm^*.b.u).$

Notice that $T \cup V0 \subset A1$. There are well-known algorithms to determine the nullable nonterminals and the productive ones (cf. [6], Lemma 4.1 and Theorem 4.3]).

Both *frk* and *brk* are relations on the set A^*, whereas *rk* of definition 4(0) is a relation on $(A^*; T^k)$. The relations *frk* and *brk* also differ from relation *rk* in two other aspects. They allow more symbol deletions: both allow deletion of productive symbols at the end of the string and of nullable symbols not in the beginning. On the other hand, each has its own condition under which productive rewritings are allowed. Relation *frk* on A^* is defined by

(3) $\quad frk.x.y \;\equiv$

$$(\exists b \subset A1 :: x = y;b)$$

$$\vee\ (\exists r,t \in A^*,\ c \in V0 :: r \neq \varepsilon \ \wedge\ x = r;c;t \ \wedge\ y = r;t)$$

$$\vee\ (\exists r \in A^*,\ t \in T^*,\ \langle c, w\rangle \in P ::$$

$$\quad |t| \leqslant k \ \wedge\ x = r;c;t \ \wedge\ y = r;w;t).$$

Relation *brk* on A^* is defined by

(4) $\quad brk.x.y \;\equiv$

$$(\exists b \in A1 :: x = y;b)$$

$$\vee\ (\exists r,\ t \in A^*,\ c \in V0 :: r \neq \varepsilon \ \wedge\ x = r;c;t \ \wedge\ y = r;t)$$

$$\vee\ (\exists r \in A^*,\ t \in T^*,\ \langle c, w\rangle \in P ::$$

$$\quad |x| \leqslant k+1 \ \wedge\ x = r;c;t \ \wedge\ y = r;w;t).$$

We use frk^* and brk^* to denote the reflexive transitive closures of *frk* and *brk*, respectively. We use *frk*-derivations and *brk*-derivations completely analogous to the *rk*-derivations introduced after definition 4(0).

For a given string x, the set of strings y with $brk^*.x.y$ can effectively be computed. In fact, the lengths of the intermediate strings is bounded by the maximum of $|x|$

and $k+j$ where j is the maximal length of a production. Therefore, the goal of the present section is to prove

(5) **Theorem.** *For $x \in (A^*;T^k)$ and $y \in (A;T^k)$ we have*

$$rk^*.x.y \;\equiv\; brk^*.x.y.$$

Remark. Since it is only an auxiliary concept, we do not mention relation frk^* in this theorem. Relation frk^* has been introduced above, since it plays a role in the concept formation and in the proof of the theorem. □

Proof. It follows from the definitions (3) and 4(0) that relation frk^* is a weakening of rk^* in the sense that

$$(\forall x,y \in (A^*;T^k) :: rk^*.x.y \;\Rightarrow\; frk^*.x.y).$$

It follows from (3) and (4) that frk^* is also a weakening of brk^* in the sense that

$$(\forall x,y \in A^* :: brk^*.x.y \;\Rightarrow\; frk^*.x.y).$$

Therefore, using $(A;T^k) \subset (A^*;T^k) \subset A^*$, we see that it suffices to prove the following reverse implications:

(6) $$(\forall x \in (A^*;T^k), y \in (A;T^k) :: frk^*.x.y \;\Rightarrow\; rk^*.x.y),$$

(7) $$(\forall x \in A^*, y \in (A;T^k) :: frk^*.x.y \;\Rightarrow\; brk^*.x.y).$$

Since formula (6) cannot serve as an induction hypothesis, it is generalised in Lemma (8) below. In fact, formula (6) follows from (8) by taking $u := \varepsilon$. Formula (7) is proved in Lemma (10) below. □

(8) **Lemma.** *Let $x \in A^*$ and $y \in (A;T^k)$ with $frk^*.x.y$. Then*

$$(\forall u \in T^* :: (x;u) \in (A^*;T^k) \;\Rightarrow\; rk^*.(x;u).y).$$

Proof. We use induction on the length of the frk-derivation from x to y. In the base case $x = y$ it suffices to observe that relation rk^* allows the deletion of all symbols of u, since $y \in (A;T^k)$ and $u \in T^*$.

Now, let $x \neq y$. In the frk-derivation from x to y, we may assume that the deletion of nullable nonterminals is performed as late as possible and then from right to left. Let $z \in A^*$ is the first intermediate string of this frk-derivation from x to y. We then have $frk.x.z$ and $frk^*.z.y$ and, by induction,

(9) $$(\forall v \in T^* :: (z;v) \in (A^*;T^k) \;\Rightarrow\; rk^*.(z;v).y).$$

Now let $u \in T^*$ be such that $(x;u) \in (A^*;T^k)$. In view of $frk.x.z$ and definition (3) we distinguish three cases.

We first consider the deletion of a productive symbol at the end of x. More precisely, we assume that $x = z;b$ with $b \in A1$. If $b \in T$ then $b;u \in T^*$ and we have $(z;b;u) = (x;u) \in (A^*;T^k)$; formula (9) yields $rk^*.(z;b;u).y$ and hence $rk^*.(x;u).y$, as required. If $b \in V \cap A1$ then $u \in T^k$. It then follows from (2) and Corollary 5(6) that $rk^*.(b;u).v$ for some string $v \in T^k$. This implies that $rk^*.(x;u).(z;v)$. Using (9) and transitivity of rk^*, we get $rk^*.(x;u).y$.

We next consider a productive rewriting of x. So, assume that $x = r;c;t$ and $z = r;w;t$ with $r \in A^*$, $t \in T^*$, $\langle c, w\rangle \in P$, and $|t| \leqslant k$. We have $|t;u| \geqslant k$, so we can choose a prefix v of u with $|t;v| = k$. We then have $rk^*.(x;u).(x;v)$ and $rk.(x;v).(z;v)$, and $rk^*.(z;v).y$ from (9). This implies $rk^*.(x;u).y$.

The third possibility is the deletion of a nullable symbol. So we have $x = r;c;t$ and $z = r;t$ with $r \in A^*$, $r \neq \varepsilon$, $t \in A^*$, and $c \in V0$. Recall that, in the *frk*-derivation from x to y, the deletion of nullable nonterminals is performed as late as possible and then from right to left. This implies that the derivation from z to y does not contain deletions of productive or nullable symbols from substring t of $x = r;c;t$, and also no rewritings of nonterminals from t. Therefore, t survives as a suffix of y. Since $r \neq \varepsilon$ and there is an *frk*-derivation from $r;t$ to y in which t survives, and since $y \in (A;T^k)$, it follows from (3) that $|t| \leqslant k$. Since t is a suffix of y, it follows that $t \in T^*$. Again, we can choose a prefix v of u such that $(t;v) \in T^k$. Since $c \in V0$, we have $rk^*.(c;t;v).(t;v)$ by Corollary 5(6), and hence $rk^*.(x;v).(z;v)$. We also have $rk^*.(x;u).(x;v)$ and $rk^*.(z;v).y$ from (9). This implies $rk^*.(x;u).y$. □

(10) **Lemma.** *Let $x \in A^*$ and $y \in (A;T^k)$ with $frk^*.x.y$. Then $brk^*.x.y$.*

Proof. Again, we use induction on the length of the *frk*-derivation from x to y. The base case $x = y$ is trivial. So, let $x \neq y$. In the *frk*-derivation from x to y we may assume that the productive rewritings occur as late as possible. Let $z \in A^*$ be the first intermediate string. We have $frk.x.z$ and, by induction, $brk^*.z.y$. If the *frk*-step from x to z is a *brk*-step then $brk^*.x.y$ and we are done.

It remains to consider an *frk*-step that is not a *brk*-step. So it is a rewriting and there exist $r \in A^*$, $t \in T^*$, and $\langle c, w\rangle \in P$ with

$$x = r;c;t \ \wedge \ z = r;w;t \ \wedge \ |t| \leqslant k.$$

Since $\neg brk.x.z$, we have $|x| \geqslant k+2$ and, hence $|r;c| \geqslant 2$ and $r \neq \varepsilon$. Again using $\neg brk.x.z$, we get $w \neq \varepsilon$, for otherwise the productive rewriting can be regarded as the deletion of a nullable symbol. It follows that $|z| \geqslant k+2$.

It follows from (4) that the *brk*-derivation from z to y begins with $|z|-(k+1)$ symbol deletions. Since, in the *frk*-derivation from x to y, the productive rewritings occur as late as possible, the *brk*-derivation from z to y does not contain symbol deletions from r and t. It follows that the *brk*-derivations from z to y begins with $|z|-(k+1)$ symbol deletions from the substring w of $z = r;w;t$. Now we observe that

$$|w| = |z| - |r;t| = |z| - (|x| - 1) \leqslant |z| - (k+1).$$

Therefore, $|w| = |z| - (k+1)$ and the *frk*-derivation from z to y begins with the stepwise deletion of string w, followed by a *brk*-derivation from $(r;t)$ to y. Recall that $w \neq \varepsilon$. If $t \neq \varepsilon$ then the stepwise deletion of w implies that $rm^*.w.\varepsilon$ and hence that $c \in V0$; this implies $brk.x.(r;t)$. If $t = \varepsilon$, the stepwise deletion of w implies that $c \in A1$ and again $brk.x.(r;t)$. In either case, it follows that $brk^*.x.y$. □

Remark. The global structure of the proofs of Theorem (5) and Lemmas (8) and (10) is quite satisfactory. The details are somewhat messy, but that may be due to the accumulated complexity of the definitions (1), (2) and (4). □

10. Concluding remarks

We regard our presentation of LR-parsing as a derivation, since, after the initial choices 4(0) and 4(1), there is essentially only one path forward and this path leads to the programs of 4(11) and 7(17). The algorithm of 7(17) is a direct descendant of Knuth's second method for testing the LR(k) condition (cf. [7, pp. 615–618]).

Distinctive features are that we have isolated the finite automaton and the stack of item sets in the optimisation treated in Section 7, and that we have isolated the delicacies of nullable nonterminals in function *Rks* which is determined in Section 9.

Another distinctive feature is that our proofs are virtually complete whereas Knuth only gives a recipe. We hope that these features form a sufficient justification for writing a heavy paper about a well-known algorithm that was treated in four pages, 26 years ago. A related analysis of Earley's algorithm (cf. [5]), is given in [4].

Acknowledgement

The present version is the result of a major revision of a version from 1989. The revision was prompted and assisted by criticisms and suggestions of H. Doornbos, a referee, J. Jongejan and J.E. Jonker.

References

[1] J. de Bakker, *Mathematical Theory of Program Correctness* (Prentice-Hall, Englewood Cliffs, NJ, 1980).
[2] E.W. Dijkstra, *A Discipline of Programming* (Prentice-Hall, Englewood Cliffs, NJ, 1976).
[3] E.W. Dijkstra and C.S. Scholten, *Predicate Calculus and Program Semantics* (Springer, Berlin, 1990).
[4] H. Doornbos, A simplification of Earley's algorithm, Computing Science Notes CS 9102, Groningen, Netherlands (1991).
[5] J. Earley, An efficient context-free parsing algorithm, *Comm. ACM* **13** (1970) 94–102.
[6] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison-Wesley, Reading, MA, 1979).
[7] D.E. Knuth, On the translation of languages from left to right, *Inform. Control* **8** (1965) 607–639.